



二维数组的花式遍历技巧

Stars 109k B站 @labuladong 配套PDF和插件 下载 打卡挑战 报名 精品课程 查看



微信搜一搜

labuladong公众号

通知：数据结构精品课 V1.7 持续更新中；B 站可查看 核心算法框架系列视频。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	151. Reverse Words in a String	151. 翻转字符串里的单词	●
-	48. Rotate Image	48. 旋转图像	●
-	54. Spiral Matrix	54. 螺旋矩阵	●
-	59. Spiral Matrix II	59. 螺旋矩阵 II	●
-	-	剑指 Offer 29. 顺时针打印矩阵	●

有不少读者说，看过很多公众号历史文章之后，掌握了框架思维，可以解决大部分有套路框架可循的题目。

但是框架思维也不是万能的，有一些特定技巧呢，属于会者不难，难者不会的类型，只能通过多刷题进行总结和积累。

那么本文我分享一些巧妙的二维数组的花式操作，你只要有个印象，以后遇到类似题目就不会懵圈了。

顺/逆时针旋转矩阵

对二维数组进行旋转是常见的笔试题，力扣第 48 题「[旋转图像](#)」就是很经典的一道：

48. 旋转图像

labuladong 题解

思路

难度 中等

👍 1077

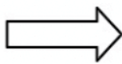


给定一个 $n \times n$ 的二维矩阵 `matrix` 表示一个图像。请你将图像顺时针旋转 90 度。

你必须在 **原地** 旋转图像，这意味着你需要直接修改输入的二维矩阵。请**不要** 使用另一个矩阵来旋转图像。

示例：

5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16



15	13	2	5
14	3	4	1
12	6	8	9
16	7	10	11

输入: `matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]`

输出: `[[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]`

题目很好理解，就是让你将一个二维矩阵顺时针旋转 90 度，**难点在于要「原地」修改**，函数签名如下：

```
void rotate(int[][] matrix)
```

如何「原地」旋转二维矩阵？稍想一下，感觉操作起来非常复杂，可能要设置巧妙的算法机制来「一圈一圈」旋转矩阵：

5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16

但实际上，这道题不能走寻常路，在讲巧妙解法之前，我们先看另一道谷歌曾经考过的算法题热身：

给你一个包含若干单词和空格的字符串 `s`，请你写一个算法，原地反转所有单词的顺序。

比如说，给你输入这样一个字符串：

```
s = "hello world labuladong"
```

你的算法需要原地反转这个字符串中的单词顺序：

```
s = "labuladong world hello"
```

常规的方式是把 `s` 按空格 `split` 成若干单词，然后 `reverse` 这些单词的顺序，最后把这些单词 `join` 成句子。但这种方式使用了额外的空间，并不是「原地反转」单词。

正确的做法是，先将整个字符串 `s` 反转：

```
s = "gnodalubal dlrow olleh"
```

然后将每个单词分别反转：

```
s = "labuladong world hello"
```

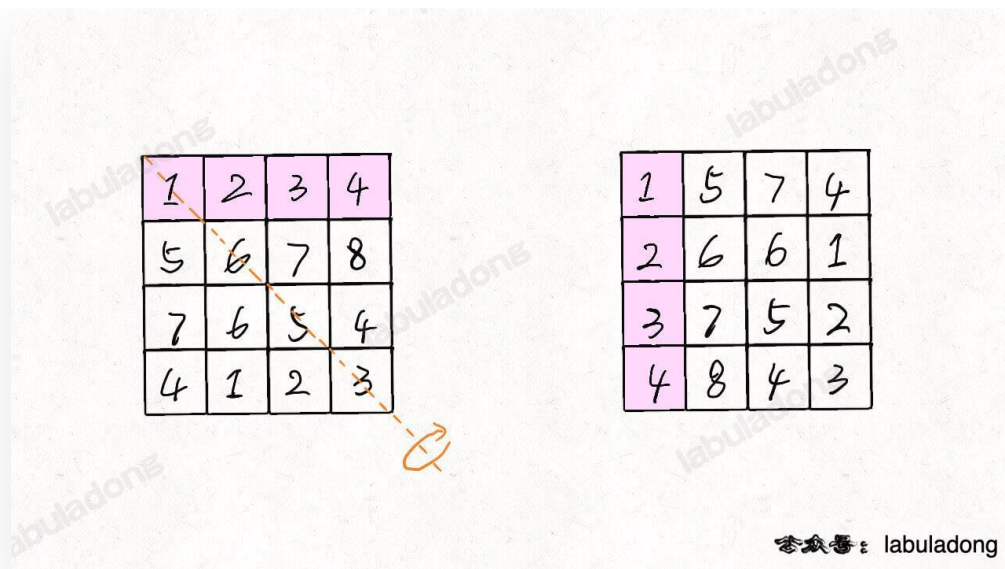
这样，就实现了原地反转所有单词顺序的目的。力扣第 151 题「[颠倒字符串中的单词](#)」就是类似的问题，你可以顺便去做一下。

我讲这道题的目的是什么呢？

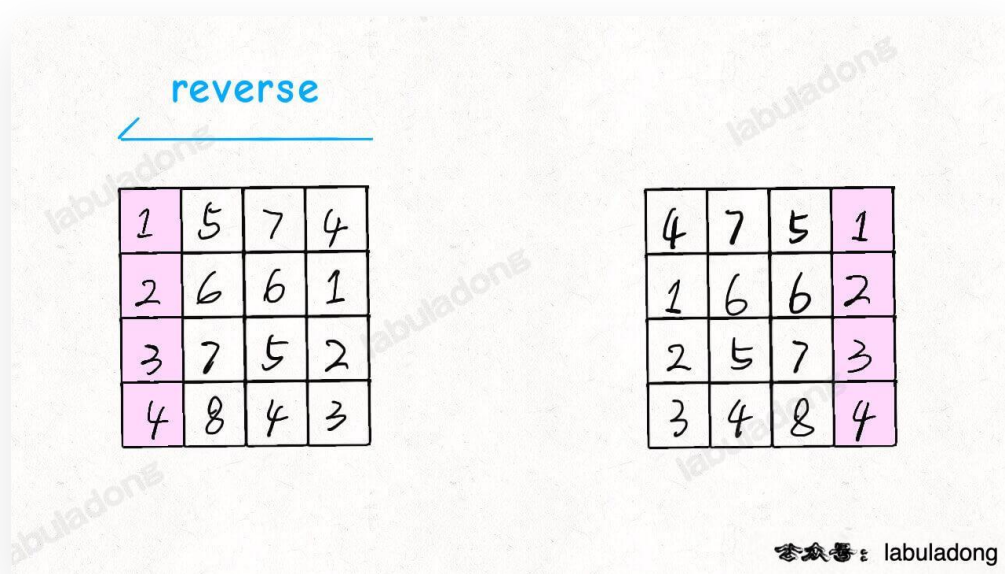
旨在说明，有时候咱们拍脑袋的常规思维，在计算机看来可能并不是最优雅的；但是计算机觉得最优雅的思维，对咱们来说却不那么直观。也许这就是算法的魅力所在吧。

回到之前说的顺时针旋转二维矩阵的问题，常规的思路就是去寻找原始坐标和旋转后坐标的映射规律，但我们是否可以让思维跳跃跳跃，尝试把矩阵进行反转、镜像对称等操作，可能会出现新的突破口。

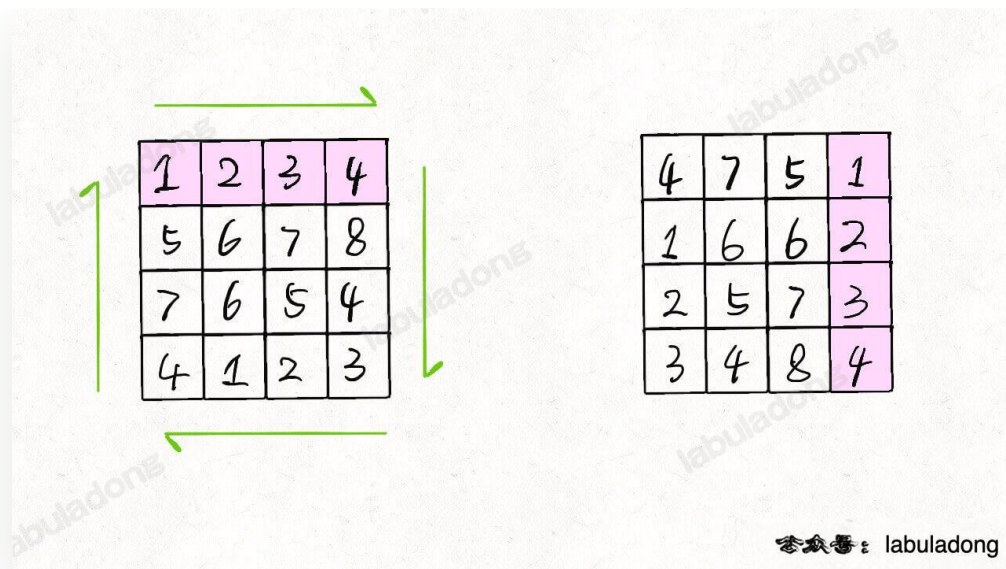
我们可以先将 `n x n` 矩阵 `matrix` 按照左上到右下的对角线进行镜像对称：



然后再对矩阵的每一行进行反转：



发现结果就是 `matrix` 顺时针旋转 90 度的结果：



labuladong

将上述思路翻译成代码，即可解决本题：

```
// 将二维矩阵原地顺时针旋转 90 度
public void rotate(int[][] matrix) {
    int n = matrix.length;
    // 先沿对角线镜像对称二维矩阵
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            // swap(matrix[i][j], matrix[j][i]);
            int temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }
    // 然后反转二维矩阵的每一行
    for (int[] row : matrix) {
        reverse(row);
    }
}

// 反转一维数组
void reverse(int[] arr) {
    int i = 0, j = arr.length - 1;
    while (j > i) {
        // swap(arr[i], arr[j]);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
        j--;
    }
}
```

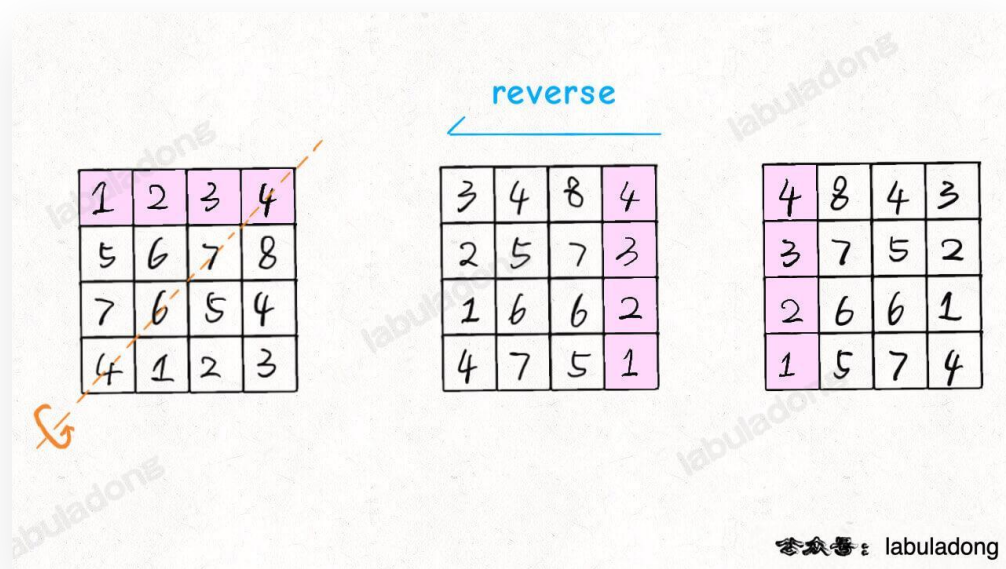

}

肯定有读者会问，如果没有做过这道题，怎么可能想到这种思路呢？

仔细想想，旋转二维矩阵的难点在于将「行」变成「列」，将「列」变成「行」，而只有按照对角线的对称操作是可以轻松完成这一点的，对称操作之后就很容易发现规律了。

既然说道这里，我们可以发散一下，如何将矩阵逆时针旋转 90 度呢？

思路是类似的，只要通过另一条对角线镜像对称矩阵，然后再反转每一行，就得到了逆时针旋转矩阵的结果：



翻译成代码如下：

```
// 将二维矩阵原地逆时针旋转 90 度
void rotate2(int[][] matrix) {
    int n = matrix.length;
    // 沿左下到右上的对角线镜像对称二维矩阵
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            // swap(matrix[i][j], matrix[n-j-1][n-i-1])
            int temp = matrix[i][j];
            matrix[i][j] = matrix[n - j - 1][n - i - 1];
            matrix[n - j - 1][n - i - 1] = temp;
        }
    }
}
```

```
// 然后反转二维矩阵的每一行
for (int[] row : matrix) {
    reverse(row);
}

void reverse(int[] arr) { /* 见上文 */ }
```

至此，旋转矩阵的问题就解决了。

矩阵的螺旋遍历

我的公众号动态规划系列文章经常需要遍历二维 `dp` 数组，但难点在于状态转移方程而不是数组的遍历，顶多就是倒序遍历。

但接下来我们讲一下力扣第 54 题「螺旋矩阵」，看一看二维矩阵可以如何花式遍历：

54. 螺旋矩阵

labuladong 题解

思路

难度 中等

👍 922

★

📄

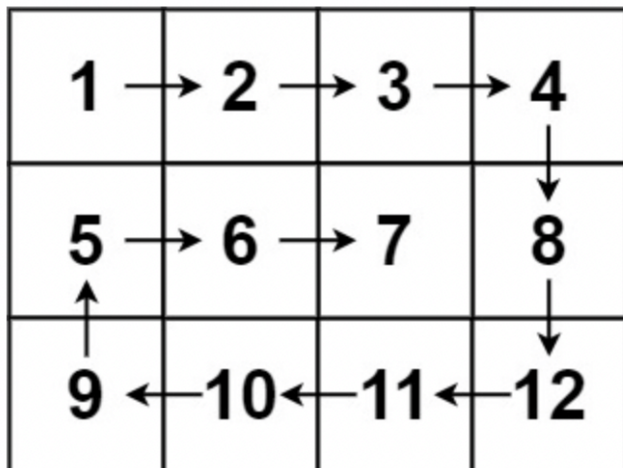
🔍

🔔

📖

给你一个 `m` 行 `n` 列的矩阵 `matrix`，请按照顺时针螺旋顺序，返回矩阵中的所有元素。

示例：



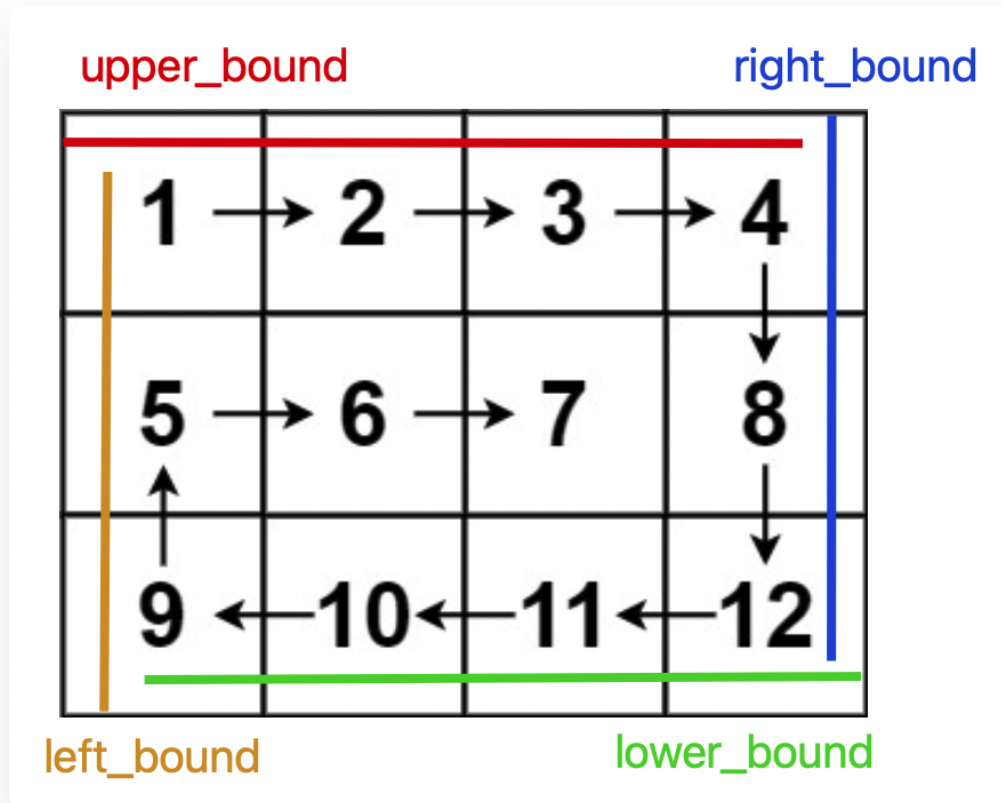
输入: `matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]`

输出: `[1,2,3,4,8,12,11,10,9,5,6,7]`

函数签名如下：


```
List<Integer> spiralOrder(int[][] matrix)
```

解题的核心思路是按照右、下、左、上的顺序遍历数组，并使用四个变量圈定未遍历元素的边界：



随着螺旋遍历，相应的边界会收缩，直到螺旋遍历完整个数组：


```

        right_bound--;
    }

    if (upper_bound <= lower_bound) {
        // 在底部从右向左遍历
        for (int j = right_bound; j >= left_bound; j--) {
            res.add(matrix[lower_bound][j]);
        }
        // 下边界上移
        lower_bound--;
    }

    if (left_bound <= right_bound) {
        // 在左侧从下向上遍历
        for (int i = lower_bound; i >= upper_bound; i--) {
            res.add(matrix[i][left_bound]);
        }
        // 左边界右移
        left_bound++;
    }
}
return res;
}

```

力扣第 59 题「螺旋矩阵 II」也是类似的题目，只不过是反过来，让你按照螺旋的顺序生成矩阵：

59. 螺旋矩阵 II

labuladong 题解

思路

难度 中等

👍 538

☆

📄

🗨

🔔

💬

给你一个正整数 n ，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的 $n \times n$ 正方形矩阵 `matrix`。

示例 1：

1	→	2	→	3
8	→	9		↓
↑				↓
7	←	6	←	5

输入：n = 3

输出：[[1,2,3],[8,9,4],[7,6,5]]

函数签名如下：

```
int[][] generateMatrix(int n)
```

有了上面的铺垫，稍微改一下代码即可完成这道题：

```
int[][] generateMatrix(int n) {
    int[][] matrix = new int[n][n];
    int upper_bound = 0, lower_bound = n - 1;
    int left_bound = 0, right_bound = n - 1;
    // 需要填入矩阵的数字
    int num = 1;

    while (num <= n * n) {
        if (upper_bound <= lower_bound) {
            // 在顶部从左向右遍历
            for (int j = left_bound; j <= right_bound; j++) {
```

```

        matrix[upper_bound][j] = num++;
    }
    // 上边界下移
    upper_bound++;
}

if (left_bound <= right_bound) {
    // 在右侧从上向下遍历
    for (int i = upper_bound; i <= lower_bound; i++) {
        matrix[i][right_bound] = num++;
    }
    // 右边界左移
    right_bound--;
}

if (upper_bound <= lower_bound) {
    // 在底部从右向左遍历
    for (int j = right_bound; j >= left_bound; j--) {
        matrix[lower_bound][j] = num++;
    }
    // 下边界上移
    lower_bound--;
}

if (left_bound <= right_bound) {
    // 在左侧从下向上遍历
    for (int i = lower_bound; i >= upper_bound; i--) {
        matrix[i][left_bound] = num++;
    }
    // 左边界右移
    left_bound++;
}
}
return matrix;
}

```

至此，两道螺旋矩阵的题目也解决了。

以上就是遍历二维数组的一些技巧，其他数组技巧可参见之前的文章 [前缀和数组](#)，[差分数组](#)，[数组双指针算法集合](#)，链表相关技巧可参见 [单链表六大算法技巧汇总](#)。

最后打个广告，我亲自制作了一门 [数据结构精品课](#)，以视频课为主，手把手带你实现常用的数据结构及相关算法，旨在帮助算法基础较为薄弱的读者深入理解常用数据结构的底层原理，在算法学习中少走弯路。

► 引用本文的题目

《labuladong 的算法小抄》已经出版，关注公众号查看详情；后台回复关键词「进群」可加入算法群；回复「PDF」可获取精华文章 PDF：



微信搜一搜

Q labuladong 公众号

共同维护高质量学习环境，评论礼仪[见这里](#)，违者直接拉黑不解释

28 Comments - powered by utteranc.es

FredGoo commented on Feb 13, 2022

48题的

```
// 将二维矩阵原地顺时针旋转 90 度
public void rotate(int[][] matrix) {
    int n = matrix.length;
    // 先沿对角线镜像对称二维矩阵
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
```

